Natural Language Processing: Part II
Overview of Natural Language Processing (L90): Part III/ACS

2017, 12 Lectures, Michaelmas Term (these notes, lectures 9 and 11)
October 27, 2017
Ann Copestake (aac@cl.cam.ac.uk)
http://www.cl.cam.ac.uk/users/aac10/

11 Lecture 11: Language generation and regeneration.

> "Generation from what?!" (attributed to Yorick Wilks)

In the lectures so far, we have concentrated on analysis, but there is also work on generating text. Although Natural Language Generation (NLG) is a recognised subfield of NLP, there are far fewer researchers working in this area than on analysis. There are some recognised subproblems within NLG, some of which will be discussed below, but it is not always easy to break it down into subareas and in some cases, systems which create text are not considered to be NLG system (e.g., when they are a component of an MT system). The main problem, as the quotation above indicates, is there is no commonly agreed starting point for NLG. The possible starting points include:

- Logical form or other sentence meaning representation.
  This is the inverse of (deep) parsing. Sometimes called *realization* when part of a bigger system (but realization is often from a syntax tree).

  One special case of this is as a component of an MT system using *syntactic transfer* or *semantic transfer*.
- Formally-defined data: databases, knowledge bases, semantic web ontologies, etc.
- Semi-structured data: tables, graphs etc.
- Numerical data: e.g., weather reports.
- User input (plus other data sources) in assistive communication.

Such systems can be contrasted with **regeneration** systems, which start from text and produce reformulated text (in the same language). Here the options include:

- Constructing coherent sentences from partially ordered sets of words: e.g., in statistical MT (although this might better be considered as a form of realization).
- Paraphrase.
- Text compression.
- Summarization (single- or multi- document).
- Article construction from text fragments (e.g., automatic construction of Wikipedia articles),
- Text simplification.

There are also mixed generation and regeneration systems. Recently, sequence-to-sequence neural models have become a very important architecture for MT and can also be used for regeneration tasks.

11.1 Tasks in generation

This categorization is broadly taken from Dale and Mellish (1998):

Content determination deciding what information to convey. This often involves selecting information from a body of possible pieces of information. e.g., weather reports. There is often a mass of information, not all of which is relevant. The content may vary according to the type of user (e.g., expert/non-expert). This step often involves domain experts: i.e., people who know how to interpret the raw data.

Discourse structuring the broad structure of the text or dialogue (Dale and Mellish refer to document structuring). For instance, scientific articles may have an abstract, introduction, methods, results, comparison, conclusion: rearranging the text makes it incoherent. In a dialogue system, there may be multiple messages to convey to the user, and the order may be determined by factors such as urgency.

Aggregation deciding how information may be split into sentence-sized chunks: i.e., finer-grained than document structuring.

Referring expression generation deciding when to use pronouns, how many modifiers to include and so on.

Lexical choice deciding which lexical items to use to convey a given concept. This may be straightforward for many applications — in a limited-domain system, there will be a preferred knowledge base to lexical item mapping, for instance, but it may be useful to vary this.

Surface realization mapping from a meaning representation for an individual sentence (or a detailed syntactic representation) to a string (or speech output). This is generally taken to include morphological generation.

Fluency ranking this is not included by Dale and Mellish but is very important in modern approaches. A large grammar will generate many strings for a given meaning representation. In fact, in most approaches, the grammar will overgenerate and produce a mixture of grammatical and ungrammatical strings. A fluency ranking component, which at its simplest is based on n-grams (compare the discussion of prediction in lecture 2), is used to rank such outputs. In the extreme case, no grammar is used, and the fluency ranking is performed on a partially ordered set of words (for instance in SMT).

When Dale and Mellish produced this classification, there was very little statistical work within NLG (work on SMT had started but wasn't very successful at that point) and NLG was essentially about limited domains. In recent years, there have been statistical approaches to all these subtasks. Most NLG systems are still limited domain however, although regeneration systems are usually not so restricted.

Many of the approaches we have described in the previous lectures can be adapted to be useful in generation. Deep grammars may be bidirectional, and therefore usable for surface realization as well as for parsing. Distributions are relevant for lexical choice: e.g., in deciding whether one word is substitutable for another. However, many practical NLG systems are based on extensive use of templates (i.e., fixed text with slots which can be filled in) and this is satisfactory for many tasks, although it tends to lead to rather stilted output.

11.2 An extended example: cricket reports

To illustrate some of these concepts further, I will use an example of constructing a textual report of a cricket match. This is partially based on Kelly's (2008) Computer Laboratory MPhil dissertation.[1]

The input Cricket produces quite rich summaries of the progress of a game, in the form of scorecards. The example below is part of a scorecard from a one day international match.

[1] Colin Kelly, Ann Copestake and Nikiforos Karamanis. Investigating Content Selection for Language Generation using Machine Learning. In: Proceedings of the 12th European Workshop on Natural Language Generation (ENLG 2009), pages 130-137. Athens, Greece, 2009.

Result India won by 63 runs						
India innings (50 overs maximum)	R	M	B	4s	6s	SR
SC Ganguly run out (Silva/Sangakarra)	9	37	19	2	0	47.36
V Sehwag run out (Fernando)	39	61	40	6	0	97.50
D Mongia b Samaraweera	48	91	63	6	0	76.19
SR Tendulkar c Chandana b Vaas	113	141	102	12	1	110.78
. . .						
Extras (lb 6, w 12, nb 7) 25						
Total (all out; 50 overs; 223 mins) 304						

The output Here is part of actual human-written report (from Wisden) roughly corresponding to some of the data shown:

> The highlight of a meaningless match was a sublime innings from Tendulkar, . . . he drove with elan to make 113 off just 102 balls with 12 fours and a six.

This requires additional knowledge compared to the scorecard: the previous results of the series which made the result here meaningless and (perhaps) knowledge of Tendulkar's play. Without such information, a possible automatic summary would be:

> India beat Sri Lanka by 63 runs. Tendulkar made 113 off 102 balls with 12 fours and a six. . . .

We will go through the stages required to achieve this.

Representing the data There are various aspects of this to consider:

- Granularity: we need to be able to consider individual (minimal?) information chunks (sometimes called 'factoids').
- Abstraction: generalize over instances.
- Faithfulness to source versus closeness to natural language?
- Inferences over data (e.g., amalgamation of scores)?
- Formalism: this might depend on the chosen methodology for realization.

A simple approach is to use a combination of predicates and constants to express the factoids, though note we need to be careful with abstraction levels:

```
name(team1/player4, Tendulkar)
balls-faced(team1/player4, 102) ...
```

Content selection There are thousands of factoids in each scorecard: we need to select the most important. For instance, the overall match result, good scores by a batsman. In this case, it is clear that Tendulkar's performance was the highlight:

```
name(team1, India)
total(team1, 304)
name(team2, Sri Lanka)
result(win, team1, 63)
name(team1/player4, Tendulkar)
runs(team1/player4, 113)
balls-faced(team1/player4, 102)
fours(team1/player4, 12)
sixes(team1/player4, 1)
```

We will discuss machine learning approaches to this step below.

Discourse structure and (first stage) aggregation This requires that we distribute data into sections and decide on overall ordering:

> Title: name(team1, India), name(team2, Sri Lanka), result(win,team1,63)
> First sentence: name(team1/player4, Tendulkar), runs(team1/player4, 113), fours(team1/player4, 12), sixes(team1/player4, 1), balls-faced(team1/player4, 102)

Sports reports often state the highlights and then describe key events in chronological order.

Predicate choice (lexical selection) If we were using a deep grammar for realization, we would need to construct an input representation that corresponded to that used by the grammar.
Mapping rules from the initial scorecard predicates:

> result(win,t1,n) $\mapsto$ _beat_v(e,t1,t2), _by_p(e,r), _run_n(r), card(r,n)
>
> name(t,C) $\mapsto$ named(t,C)

This gives:

> name(team1, India), name(team2, Sri Lanka), result(win,team1,63) $\mapsto$
> named(t1,'India'), named(t2, 'Sri Lanka'), _beat_v(e,t1,t2), _by_p(e,r), _run_n(r), card(r,'63')

Realistic systems would have multiple mapping rules. This process may require refinement of aggregation.

Generating referring expressions

> named(t1p4, 'Tendulkar'), _made_v(e,t1p4,r), card(r,'113'), run(r), _off_p(e,b), ball(b), card(b,'102'), _with_(e,f), card(f,'12'), _four_n(f), _with_(e,s), card(s,'1'), _six_n(s)

corresponds to:

> Tendulkar made 113 runs off 102 balls with 12 fours with 1 six.

which is not grammatical. So convert:

> _with_(e,f), card(f,'12'), _four_n(f), _with_(e,s), card(s,'1'), _six_n(s)

into:

> _with_(e,c), _and(c,f,s), card(f,'12'), _four_n(f), card(s,'1'), _six_n(s)

Also: '113 runs' to '113' because, in this context, it is obvious that 113 would refer to runs.

Realization and fluency ranking Produce grammatical strings in ranked order:

> Tendulkar made 113 off 102 balls with 12 fours and one six.
> Tendulkar made 113 with 12 fours and one six off 102 balls.
> ...
> 113 off 102 balls was made by Tendulkar with 12 fours and one six.

11.3 Learning from existing text

To a certain extent, it is possible to learn to generate such reports from existing pairings of data and text: here from pairings of scorecards and cricket reports, which can conveniently be downloaded. In order to do this, the initial step is to annotate the text of the reports with appropriate scorecard entities:

> The highlight of a meaningless match was a sublime innings from Tendulkar (team1 player4), ... and this time he drove with elan to make 113 (team1 player4 R) off just 102 (team1 player4 B) balls with 12 (team1 player4 4s) fours and a (team1 player4 6s) six.

- Either: annotate training set by hand (boring!)
- Or: write rules to create training set automatically, using numbers and proper names as links.

Content selection can now be treated as a classification problem: all possible factoids are derived from the data source and each is classified as in or out, based on training data. The factoids can be categorized into classes and grouped. One problem that Kelly found was that the straightforward technique returns 'meaningless' factoids, e.g. player names with no additional information about their performance. This can be avoided, but there's a danger of the approach becoming task-specific.

Discourse structuring then involves generalising over reports to see where particular information types are presented. Fluency ranking and predicate choice could straightforwardly use the collected texts. Kelly didn't do these steps but they have been done in other contexts (e.g., Wikipedia article generation).

11.4 Referring expressions

One subtopic in NLG which has been extensively investigated is generation of referring expressions: given some information about an entity, how do we choose to refer to it? This has several aspects: we have to decide whether to use ellipsis or coordination (as in the cricket example above) and which grammatical category of expression to use: pronouns, proper names or definite expressions for instance. If a pronoun is used it must be correctly resolved by the hearer: there are some bidirectional approaches to anaphora but an alternative is to generate multiple options and test them using a standard anaphora resolution algorithm as discussed in the previous lecture. Finally, if we choose to use a full noun phrase we need to consider attribute selection: that is we need to include enough modifiers to distinguish the expression from possible *distractors* in the discourse context: e.g., *the dog*, *the big dog*, *the big dog in the basket*. This last aspect has received the most attention and will be described in more detail here. The idea of a *distractor* is relevant in a number of other contexts within NLP. For instance, when setting up datasets for training and evaluating systems via question answering, it is important that the alternatives are plausible. Similarly, the task of building suitable sets of multiple-choice questions for testing humans requires appropriate alternatives to the correct answer.

Emiel Krahmer, Sebastiaan van Erg and André Verleg (2003) 'Graph-based generation of referring expressions' describe a meta-algorithm for generating referring expressions: a way of thinking about and/or implementing a variety of algorithms for generating referring expressions which had been discussed in the literature. In their approach, a situation is described by predicates in a knowledge base (KB): these can be thought of as arcs on a graph, with nodes corresponding to entities. An example from their paper is given below.

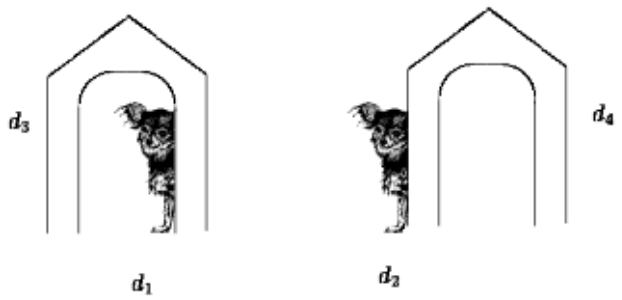


Figure 2
Another scene: Two dogs and two doghouses (from Krahmer and Theune [2002]).

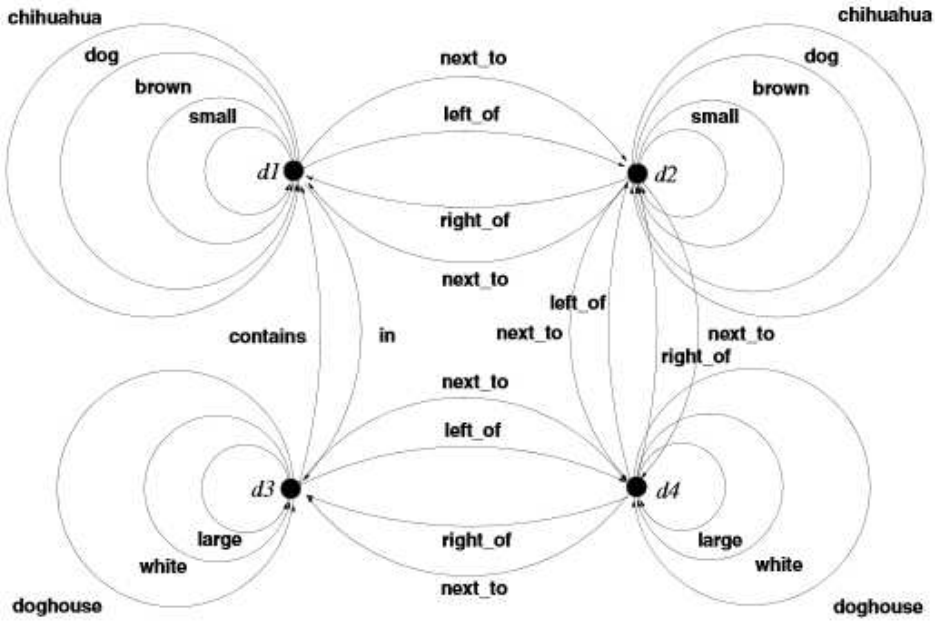


Figure 3
A graph representation of the scene in Figure 2.

A description (e.g., *the dog*, *the small brown dog*) is then a graph with unlabelled nodes: it matches the KB graph if it can be 'placed over' it (formally, this is *subgraph isomorphism*). A *distinguishing graph* is one that refers to only one entity (i.e., it can only be placed over the KB graph in one way). If we have a description that can refer to entities other than the one we want, the other entities are referred to as *distractors*. In general, there will be multiple distinguishing graphs: we're looking for the one with the lowest cost — what distinguishes algorithms is essentially their definition of cost.

The algorithm starts from the node which we wish to describe and expands the graph by adding adjacent edges (so we always have a connected subgraph). The cost function is given by a positive number associated with an edge. We want the cheapest graph which has no distractors: we explore the search space so we never consider graphs which are more expensive than the best one we have already. If we put an upper bound K on the number of edges in a distractor, the complexity is n^K (i.e., technically it's polynomial, since K is a constant, and probably it's going to be less than 5). Various algorithms then correspond to the use of different weights.

The *full brevity* algorithm is described in Dale (1992): in terms of the meta-algorithm, it's equivalent to giving each arc a weight of 1. It is guaranteed to produce the shortest possible expression (in terms of logical form rather than the string). Dale (1992) also describes a greedy heuristic, which can be emulated by assuming that the edges are ordered by discriminating power. This gives a smaller search space.

However, subsequent experiments suggest that these algorithms may not give similar results to humans. One issue is that verbosity isn't always a bad thing: we don't want the user to have to make complex inferences to determine a referent, and sometimes slightly longer phrases might be useful to reinforce a salient point. Occasionally, they even just sound better. In dialogue contexts it may be advisable to be verbose because it sounds more polite or just because it improves the user's chance of understanding a speech synthesizer. So later work on referring expressions has relaxed the requirement to minimize noun phrase modifiers.

There are also difficulties associated with the idea that the algorithm is run using KB predicates without knowledge of syntax of the natural language expressions. For example, consider the alternative terms, *earlier* and *before*. In a domain about diaries and meetings, these lexemes might be considered to map to the same KB predicate. However, they differ in their syntax:

The meeting earlier than my lunchtime appointment is cancelled.
The meeting before my lunchtime appointment is cancelled.
The earlier meeting is cancelled.
* The before meeting is cancelled.

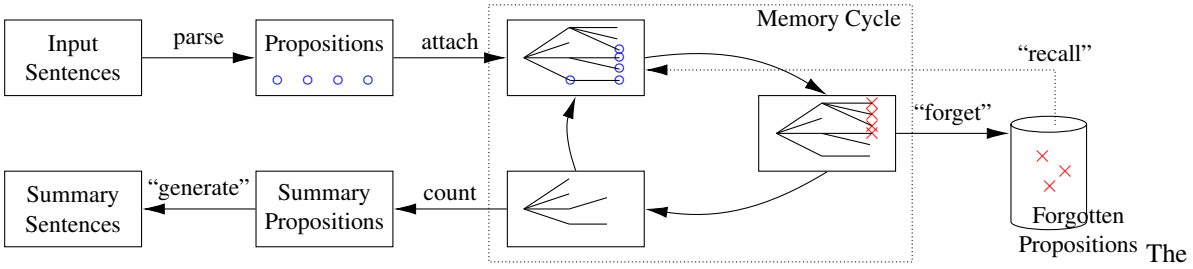
This matters, because when generating expressions where two entities are required, we need to be able to generate good referring expressions for both entities. Hence the abstraction of considering referring expressions in terms of KB predicates is not entirely satisfactory. Furthermore, referring expression generation is also needed in regeneration contexts without a limited domain and hence where there is no KB. For instance, after simplifying a text by replacing relative clauses by new sentences, it is often necessary to reformulate the referring expressions in the original text. Taken together, these issues imply that corpus-based approaches to referring expression generation may be preferable.

11.5 Summarization: a proposition-based abstractive summarizer

Summarization systems can be broadly categorized as *extractive* or *abstractive*. An extractive system chooses the 'best' sentences from a piece of text to produce the summary. For newspaper text, articles are usually written so that the first sentence provides a good summary: hence choosing the first sentence is a very good baseline. Extracted sentences may then be compressed to further shorten the output. Abstractive summarizers partially analyse the text and use that analysis to build a summary. The system outlined here (Fang and Teufel, 2014, 2016; Fang et al, 2016) produces a set of propositions.

1. Parse each input sentence, create a set of propositions.
2. Attach each proposition to a *coherence tree* representing the document as a whole, using argument overlap to decide on the attachment.
3. Provisionally 'forget' some of the existing propositions (remove them and store them).
4. Carry on to the next sentence.
5. If propositions cannot be coherently attached at phase 2, recover temporarily forgotten propositions so that the new propositions can be attached.
6. Finally: rank propositions by the number of cycles that they survived, and generate (realize) the summary from the highest ranked propositions.

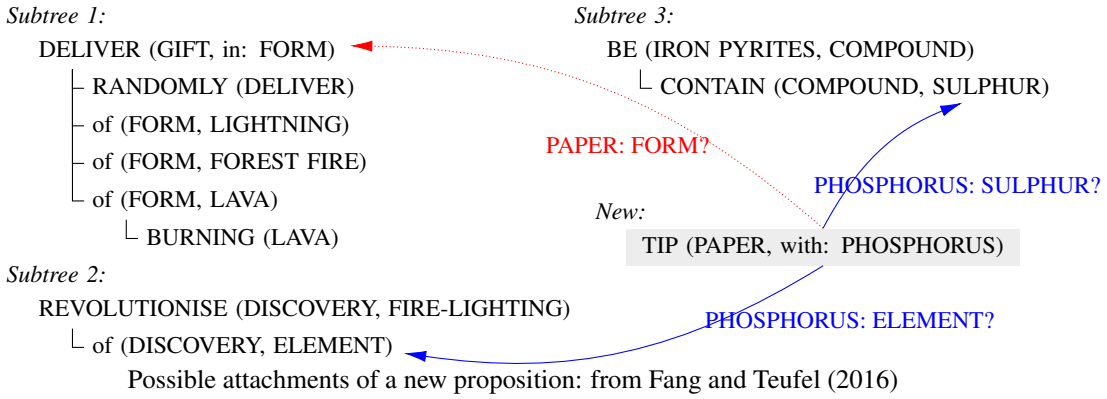
This overall architecture is illustrated below:



The architecture of the summarizer (Fang and Teufel 2016)

Fang et al (2016) demonstrate the use of a deep realizer from a bidirectional grammar to create the final output.

Proposition attachments will generally be ambiguous. An example is shown in the figure below:



Possible attachments of a new proposition: from Fang and Teufel (2016)

Fang and Teufel describe this as follows:

> Three subtrees in the working memory are shown, containing propositions that correspond to the text pieces 1) *[fire was] a gift randomly delivered in the form of lightning, forest fire or burning lava*, 2) *fire-lighting was revolutionised by the discovery of the element*, and 3) *iron pyrites, a compound that contains sulphur*, respectively. The new proposition corresponds to the text *paper tipped with phosphorus*. It can attach in subtree 2, because *phosphorus* is a kind of *element*; it can also attach in subtree 3, because both *phosphorus* and *sulphur* are chemicals.

In the limit, the attachment of the incoming propositions requires arbitrary world knowledge and reasoning. However, Fang and Teufel (2016) demonstrate the effectiveness of a WordNet-based system of deriving *lexical chains*, which I will not describe in detail here.

Evaluation of this system (e.g., Fang et al 2016) suggests it is a promising alternative to extractive summarization.

11.6 Further reading

Unfortunately the second edition of J&M has almost nothing about NLG. There is a chapter in the first edition, but it is now extremely dated. Reiter and Dale (2000) *Building Natural Language Generation Systems* is a textbook that discusses the general concepts, but is also now outdated.

Blogging Birds (`http://redkite.abdn.ac.uk/`) is a great example of a system that generates text based on data: see `http://aclweb.org/anthology/P/P13/P13-4029.pdf` for a short description.

Summarization is described in Chapter 23 of J+M.

12 Lecture 12: Recent NLP research.

I will discuss Visual Question Answering during this lecture. There are no notes: copies of slides will be made available after the lecture.